Part II – Black-Box Composition Systems
10. Finding Business Components in a Component-Based Development Process
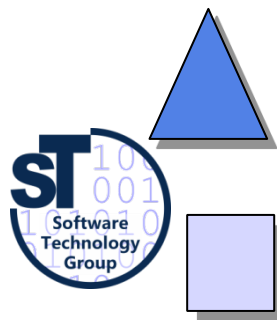
1. The UML component model

2. Business component model of the Cheesman/ Daniels process

3. Identifying business components

Prof. Dr. Uwe Aßmann

Technische Universität Dresden

Institut für Software- und Multimediatechnik

http://st.inf.tu-dresden.de

13-1.2, 24.04.13

Literature

► J. Cheesman, J. Daniels. UML Components. Addison-Wesley.

10.1 Big Objects, Business Objects, and UML Components

The Cheesman-Daniels approach identifies UML components in UML class diagrams, adding required and provided interfaces.

It describes how to transform a UML class diagram to a UML component diagram.

Natural and Dependent Types

- ➢ An object with a **natural type (entity type)** lives on its own and exists independent of context and collaborators
  - ▪ The type does not depend on other types (**independent type**)
    - . Hotel vs. HotelRoom
    - . Car vs. Screw or Motor
  - ▪ Types that depend on others are called **dependent types.**
- ▪ Role types, facet types, part types are dependent types.
- ➢ A **big object (bob)** is complex, hierarchical object with a natural type
  - • Usually, it has subobjects with dependent types, role types and others.
- ► A **business object (domain object)** is a bob with a natural type of the domain model (business model)
  - ► Usually, business objects (domain objects) are large hierarchical objects
  - ▪ They can consist of thousands of smaller objects of dependent types (part-of relation)
 - ▪ They can play many *roles* with *context-based types*
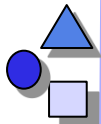
Component Specification with UML Components

- A **UML component** is a hierarchical class for big objects with *provided* and *required* interfaces (roles)
    - Provided interfaces (provided roles) use „lollipop" notation
    - Required interfaces (required roles) use „plug" notation
- UML components can specify bobs
with one natural core object and many dependent subobjects
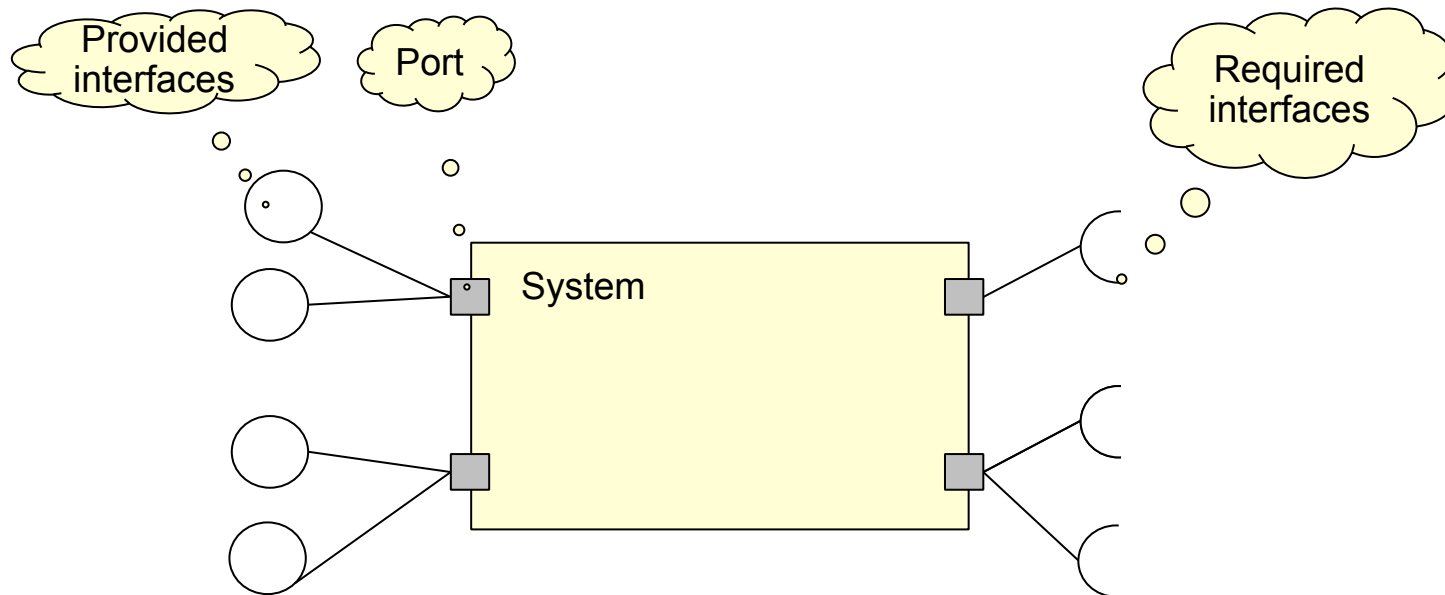

Some components are required to use specific other interfaces
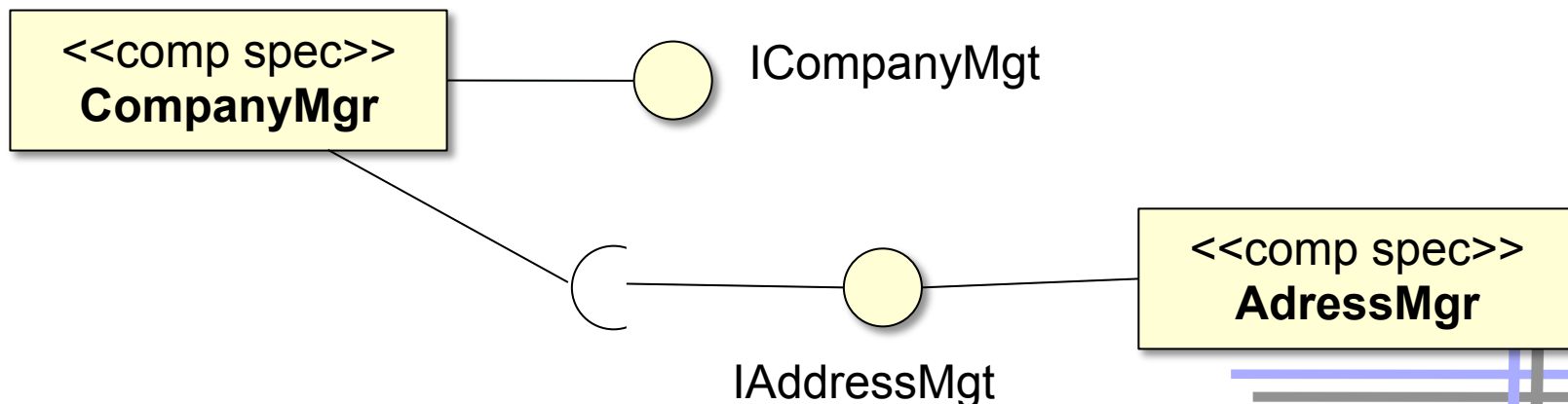
Ports of UML Components

➢ A **port** is a connection point of a UML component.

  A port has a set of roles (interfaces)

  It may be represented by a **port object (gate)**

Lollipops und Plugs (Balls and Sockets)

► For a UML component, *provided* and *required interfaces* can be distinguished

  ▪ A required interface specifies what the current class needs to execute.
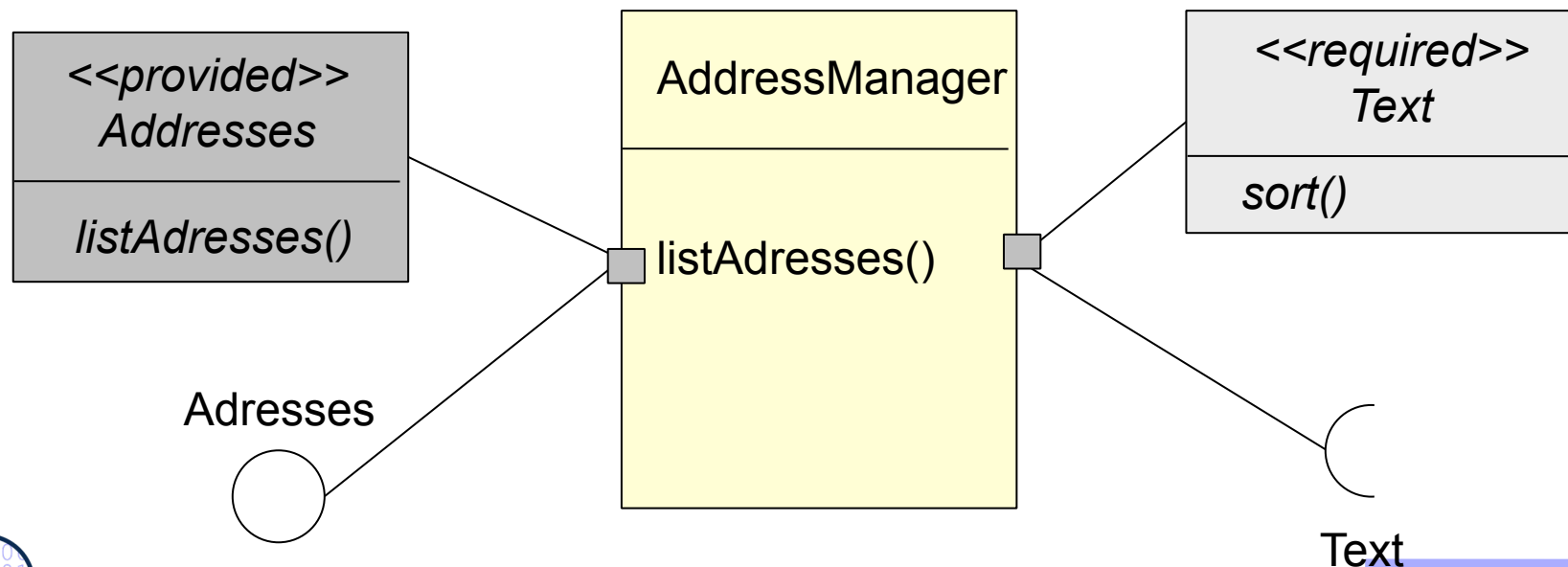
Ports

► Ports consist of **port classes** with interfaces and behavior in form of **interface automata**

- provided: normal, *offered* interface
- required: used, *necessary* interface
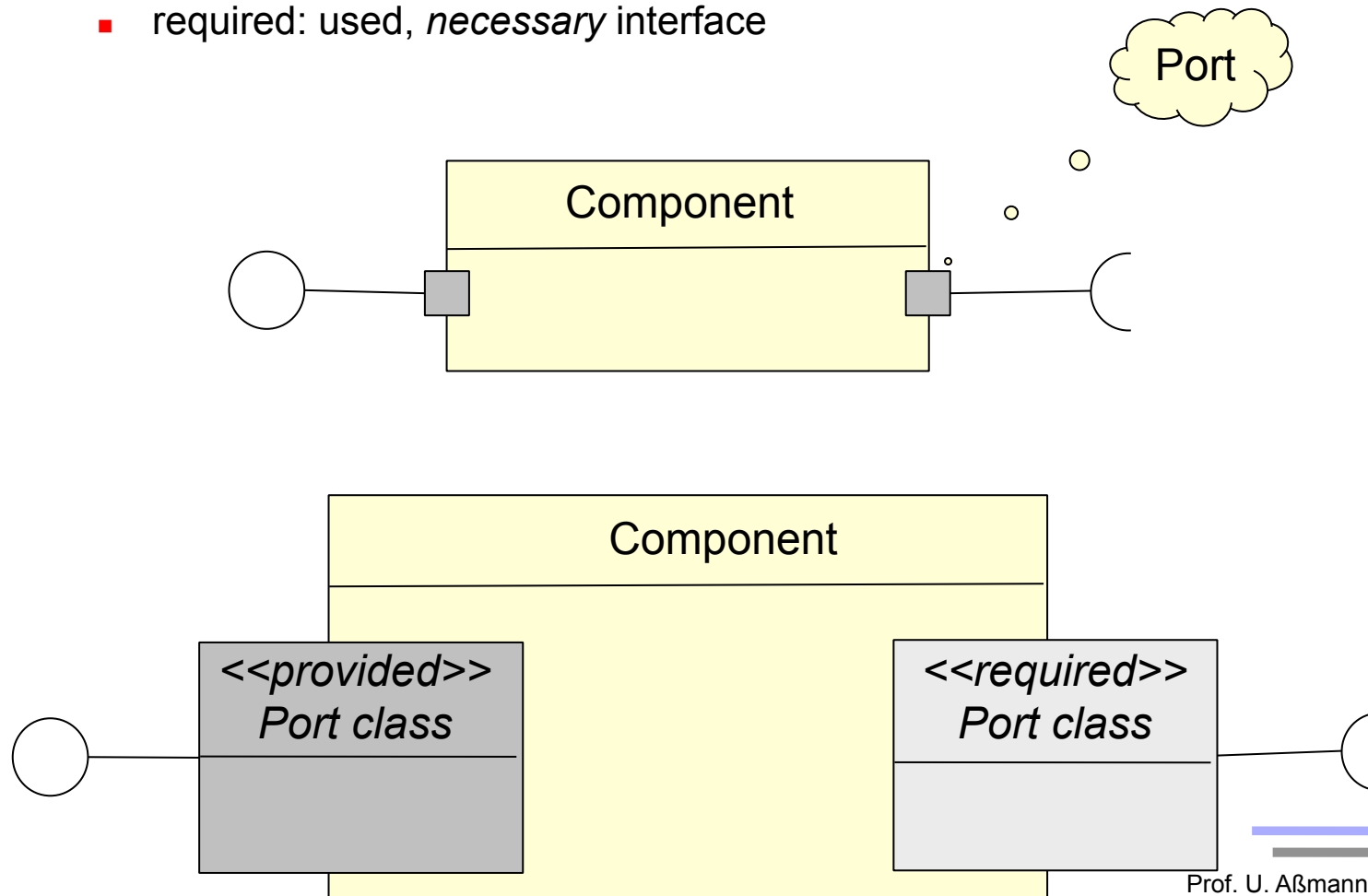
Nesting of UML Components

► UML components

- Ports are connected by *links (connections)*
- *Delegation link*: links outer and inner port
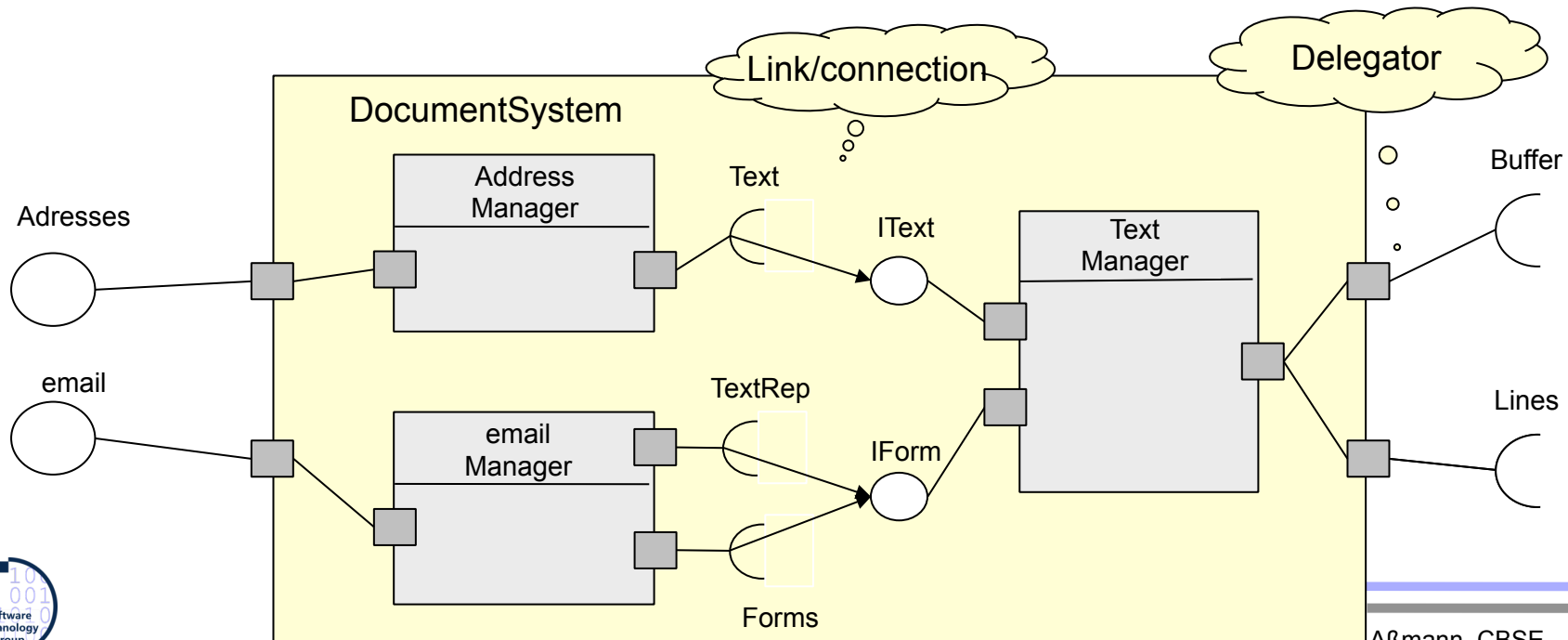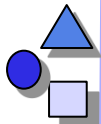
Refinement of UML Components

► UML components are nested, i.e., are bobs.
► Nesting is indicated by *aggregation* and *part-of relationship*.
► Nesting is introduced by an encapsulation operator `encapsulate`.

encapsulate

Document
System

decompose

Document System

Address
Manager

Adresses

Text

IText

Text
Manager

Buffer

email
Manager

email

TextRep

IForm

Lines

Forms
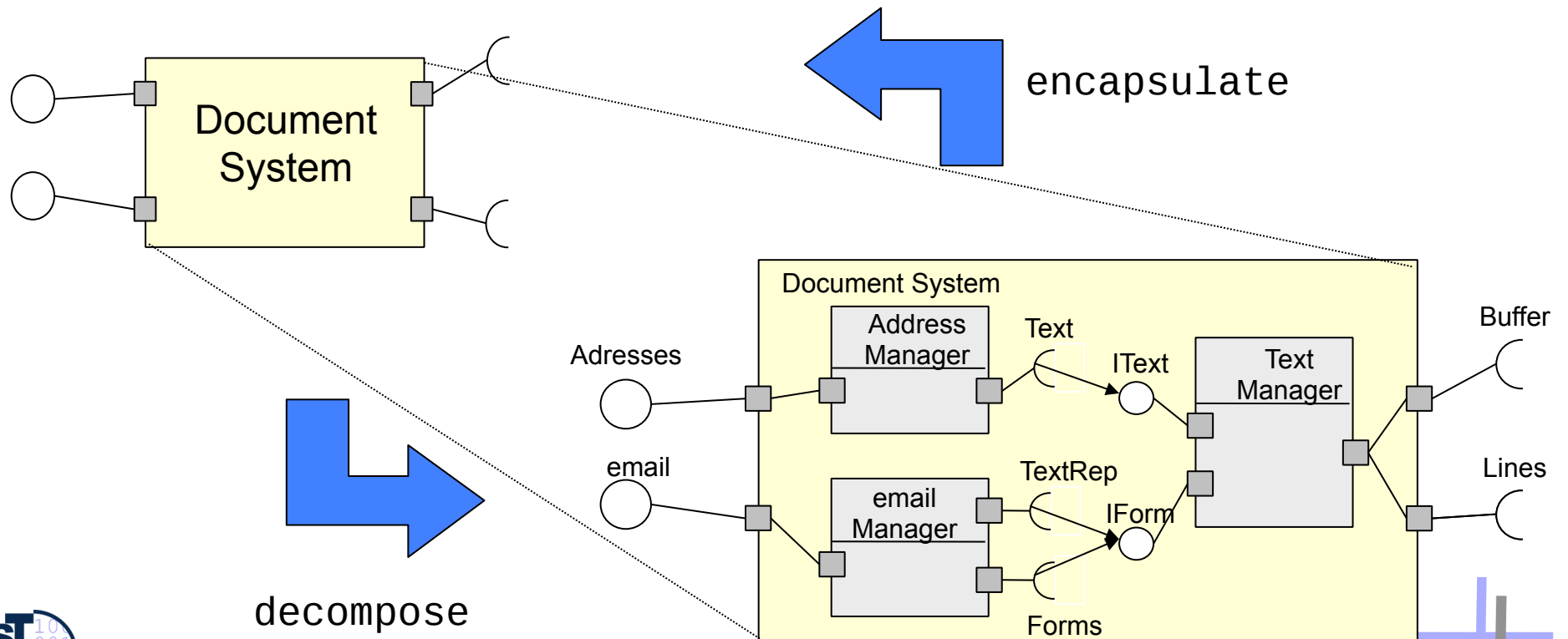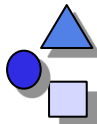
Encapsulation means Aggregation

► *Nesting* means *Aggregation*

- ▪ A UML component is a package and a façade for all subcomponents
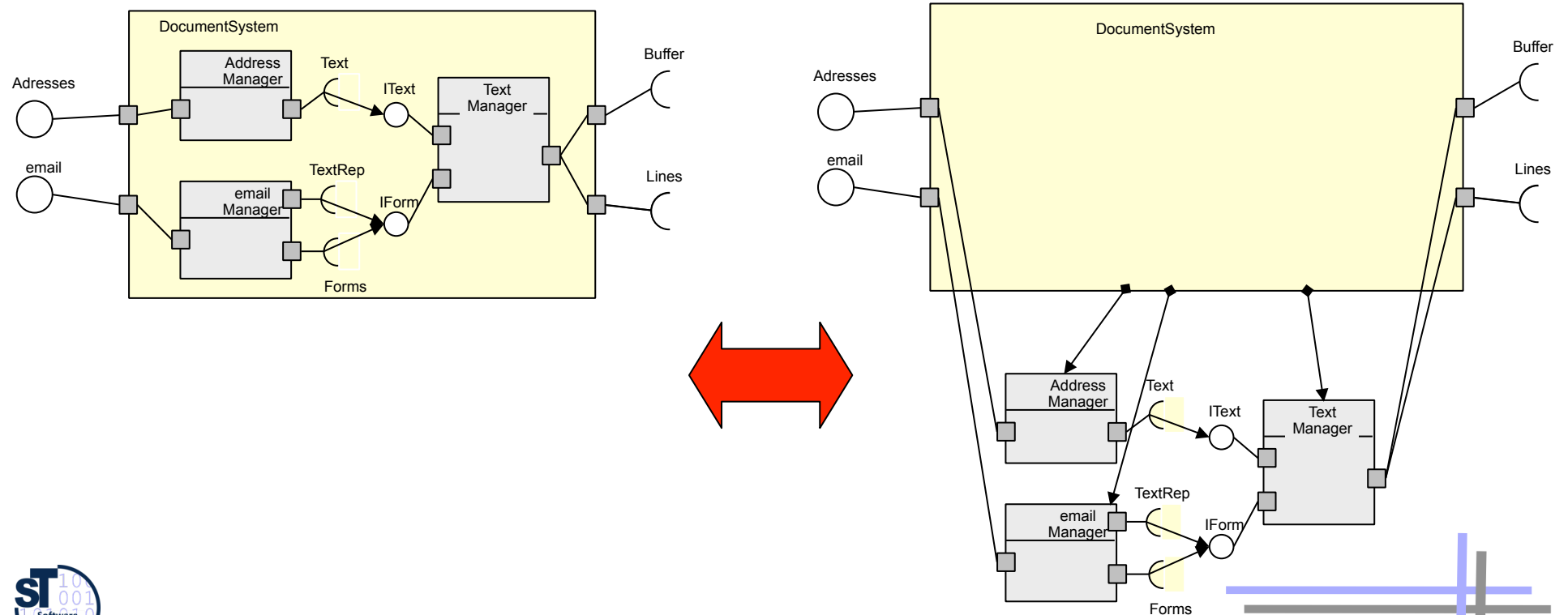
10.2 A Business Component Model

The Cheesman-Daniels process to find business components

Business Objects are Complex Objects

► In the Cheesman-Daniels component model, a **business component** consists of a set of business objects and other business components (part-of relation)

  ► The smallest component is a *business object with several provided and required interfaces*

  . The business objects are the logical entities of an application

  . Their interfaces are re-grouped on system components for good information hiding and change-oriented design

■ A business component has a specification containing all interfaces and contracts and an implementation

  ■ UML-CD are used (UML profile with stereotypes)
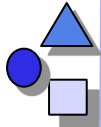
Goals of the Cheesman-Daniels Process

► The Cheesman-Daniels Process identifies UML components in UML class diagrams

  ► It bridges *domain modelling* with *use case modelling* (functional requirements)

► Steps:

  ► Find out business objects (big objects with core and subobjects) of the application

  ► Group business objects to components with required and provided interfaces, for change-oriented design and reuse

  ► Specify contracts for the components

► Be aware: the Cheesman-Daniels Process can be employed also for many other component models of this course, such as

  ► Black box component models, such as EJB, Corba, .NET

  ► Grey-box component models:

    ► Generics (e.g., class diagram templates)

    ► Fragment component models (e.g., advice groups in aspects)

    ► Class-role models

Business Component Model

► In the Cheesman-Daniels component model, a **business component** consists of a set of business objects and other business components (part-of relation)

► The smallest component is a *business object with several provided and required interfaces*

  . The business objects are the logical entities of an application

  . Their interfaces are re-grouped on system components for good information hiding and change-oriented design

■ A business component has a specification containing all interfaces and contracts and an implementation

  ■ UML-CD are used (UML profile with stereotypes)

10.3. Identifying Business Components

Identifying Business Components with the Cheesman-Daniels Process

➢ Overall development process

Requirements

Business Concept models

Specification

1) Component Identification → 2) Component Interaction

3) Component Specification

Constraints

4) Provisioning

Components

5) Assembly

Use Case models

Existing assets
Component Specs & Architectures

Test

Deployment

Simplified version of Fig. 2.1 from Cheesman/Daniels
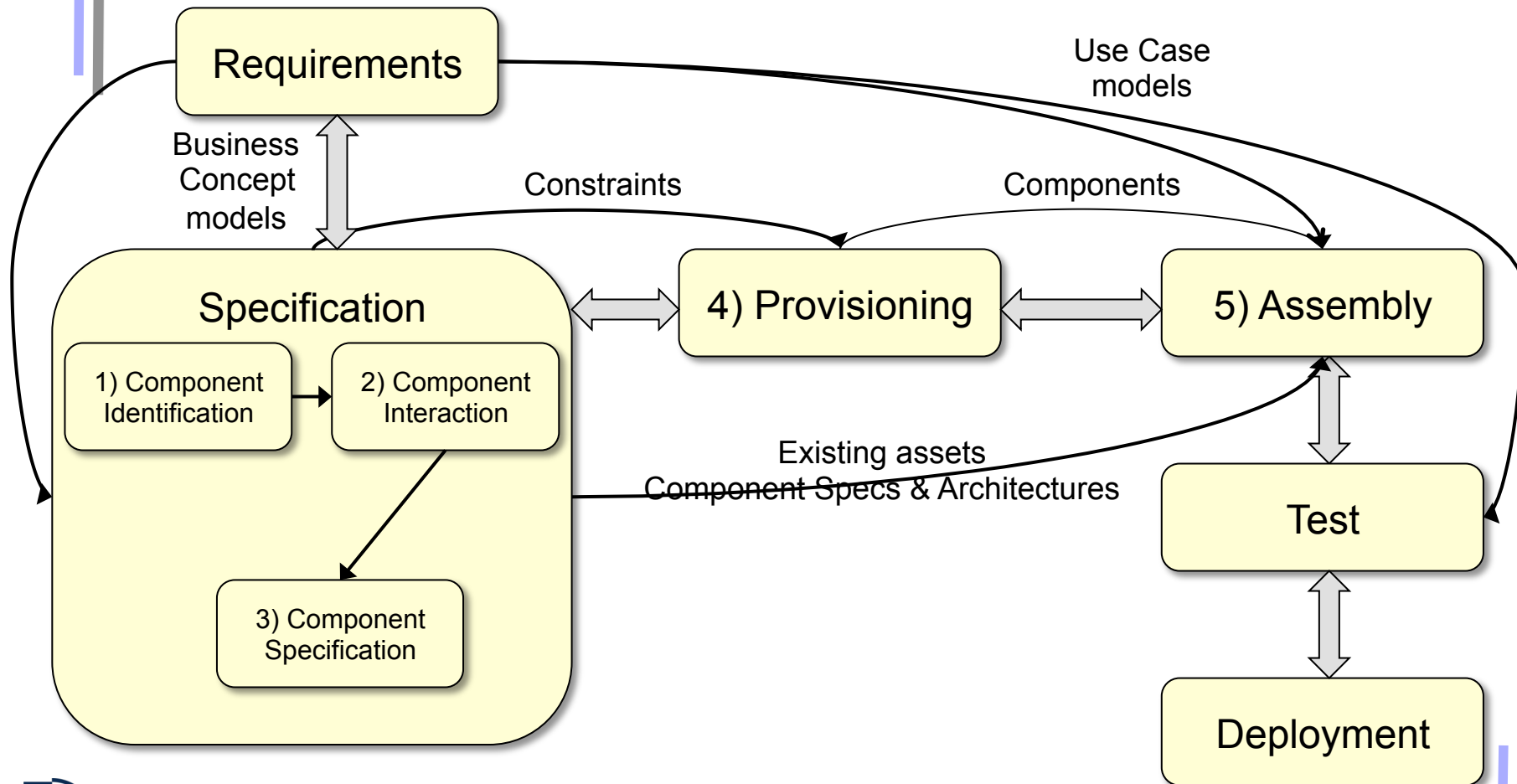
Artifacts of the Cheesman/Daniels Process

► Requirement artifacts:

- *Domain model (business concept model):* describes the business domain (application domain)
- *Use case model* (requirements model)

► System artifacts, derived from the business concept model:

- *Business type model*, class diagram derived from domain model:
    - Represents the system's perspective on the outer world (more attributes, refined class structures from the system's perspective)
- *Business object interface model*, identifies the business objects and all their interfaces
- *Business object model*, derived from the business object interface model by adding additional operations

► System component artifacts

- Component interface specifications: one contract with the client
- Component interface information model (state-based model)
- Component specifications: all interface specifications of a component plus constraints.
- Component architecture: wiring (topology) of a component net.

10.3.1 Component Identification (Step 1)

Domain Model (Business Concept Model)	Domain analysis	Use Case Model	Function analysis	Component Specification Database	Reuse analysis

Component Identification

- Select Business Types From Domain Model
- Business Type Model
- Find out Business Object Interfaces
- Business Object Interface Model

- Develop system interfaces model
- Decompose top-down
- Allocation of business object interfaces to components
- New component specifications

- Find component specifications (matchmaking)
- Reusable component specifications

Component Specifications and Architecture

Ex.: Domain Model of a Course-Management System

► Collects all concepts of the domain (aka business concept model)
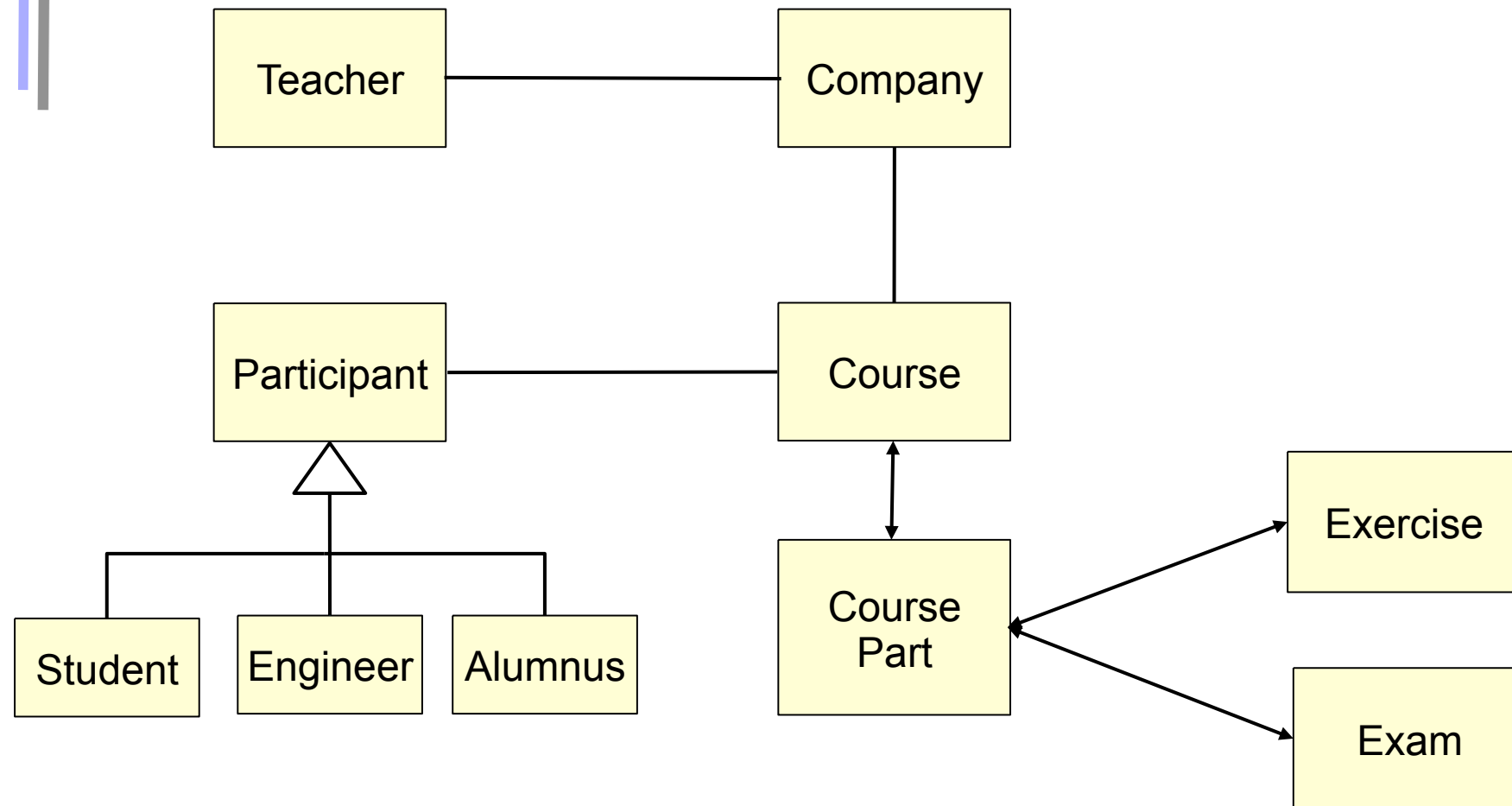
10.3.1.a) Business Type Model

► Defines system types from the domain model

  - Eliminates superfluous concepts

  - Adds more details

  - Distinguish datatypes (passive objects)
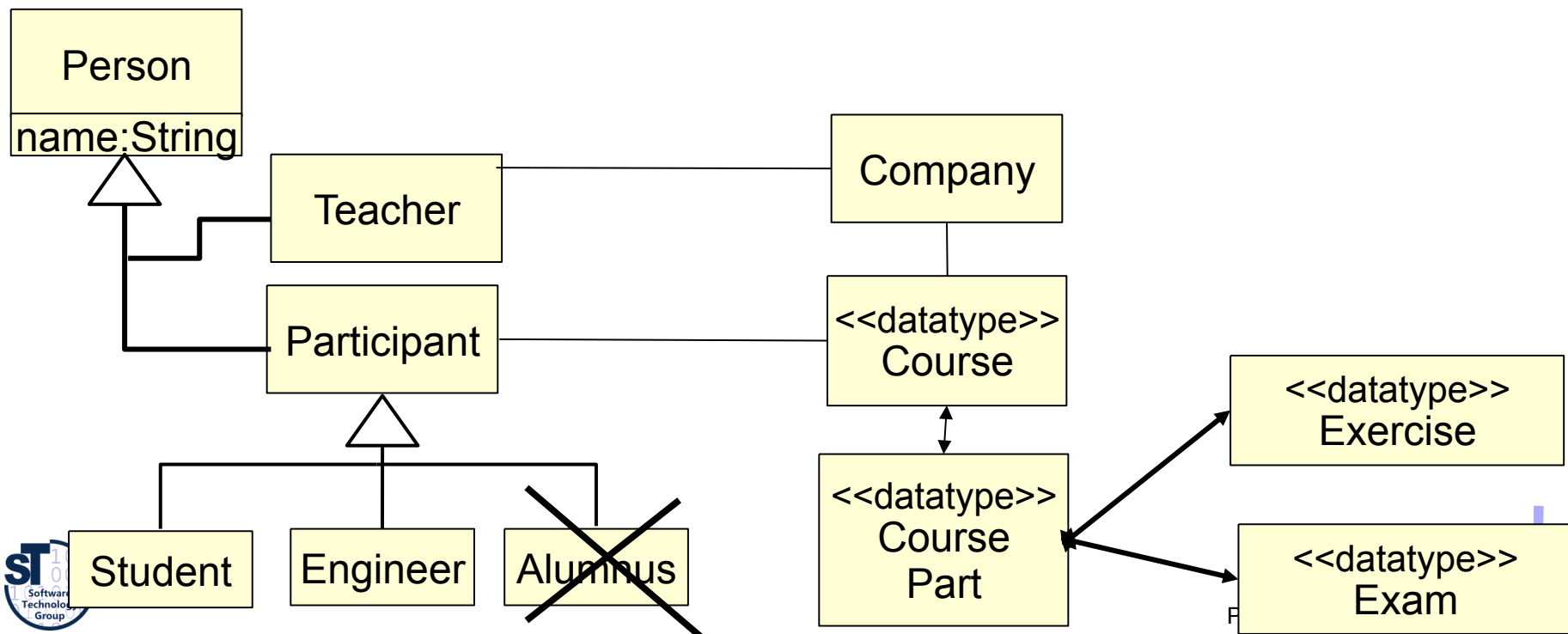
10.3.1.b) Business Object Interface Model

► Identifies business objects from the business type model

  ▪ And defines *management interfaces* for them

  ▪ Here, only Company, Course, Person are business objects, all others are dependent types

IPersonMgmt

ICompanyMgmt

ICourseMgmt

<<business object>> Person
name:String

Teacher

| <<business object>> Company |

| <<business object>> Course |

Participant

Student

Engineer

Course Part

Exercise

Exam

Prof. U. Aßmann
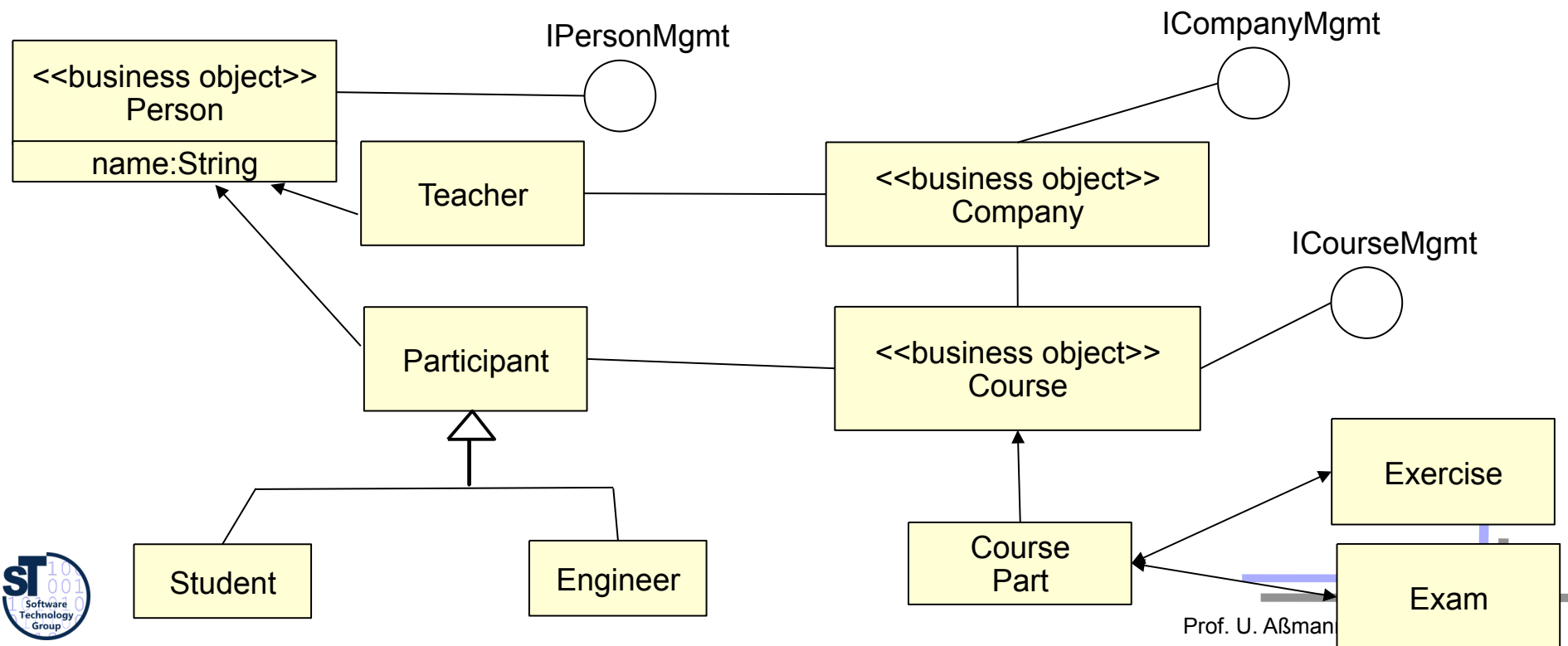
10.3.1.c) Component Identification (Version 0.1)

► Group classes and interfaces into reusable components

ICompanyMgmt

IPersonMgmt

ICourseMgmt

<<comp spec>>
Company

<<business object>>
Company

<<comp spec>>
Repository

<< business object>>
Person

name:String

Teacher

Participant

Student

Engineer

<<business object>>
Course

Course Part

Exercise

Exam
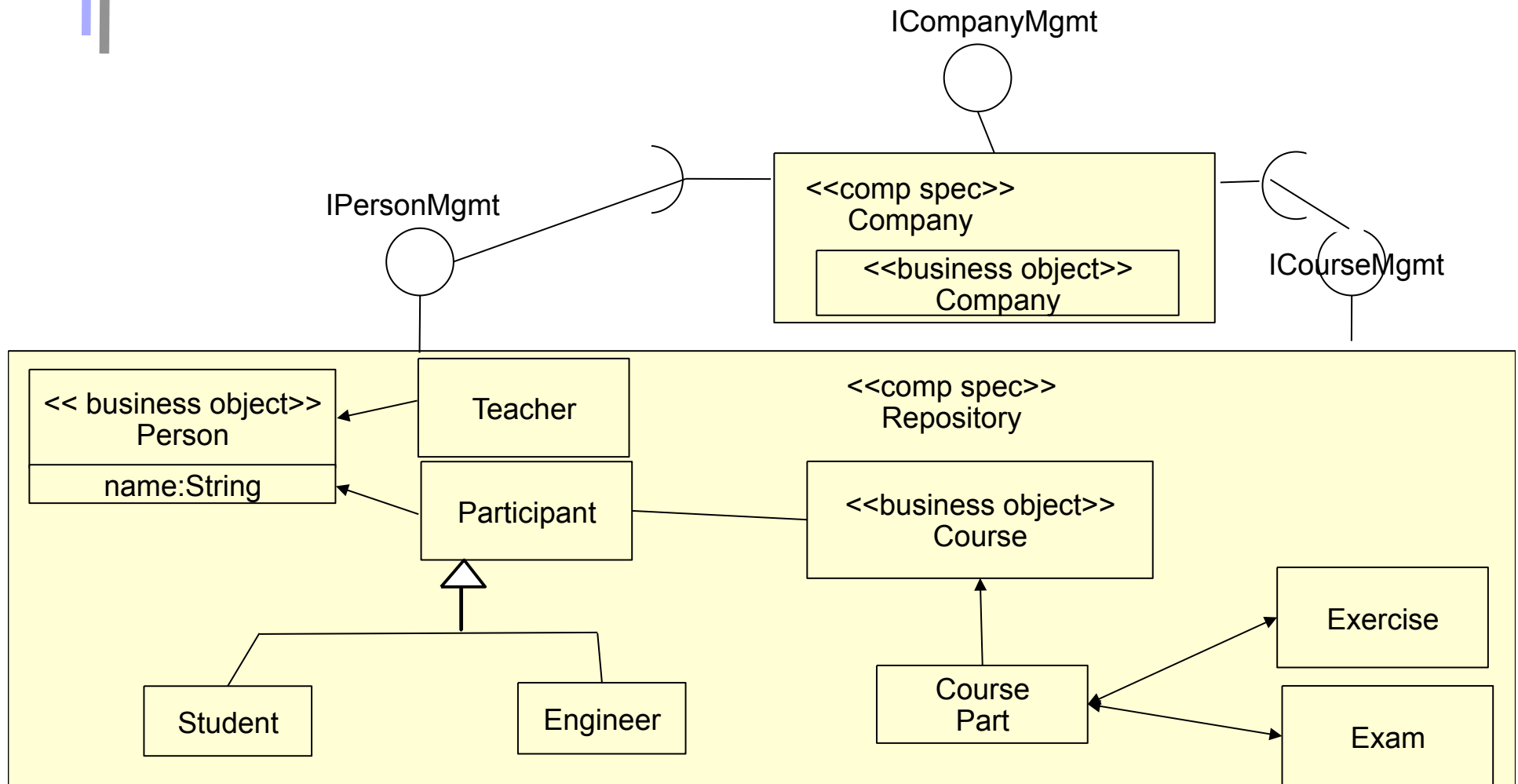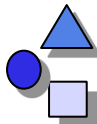
Alternative Component Identification (0.1)

► Often, classes and interfaces can be grouped in several ways into components. Goal: think about what is reusable

► Here: Person management might be reuseable, so make it a separate component

ICompanyMgmt

IPersonMgmt

<<comp spec>>
Company

<<business object>>
Company

ICourseMgmt

<<business object>>
Person

name:String

<<comp spec>>
Persons

Teacher

Participant

Student

Engineer

<<comp spec>>
Courses

<<business object>>
Course

Course
Part

Exercise

Exam
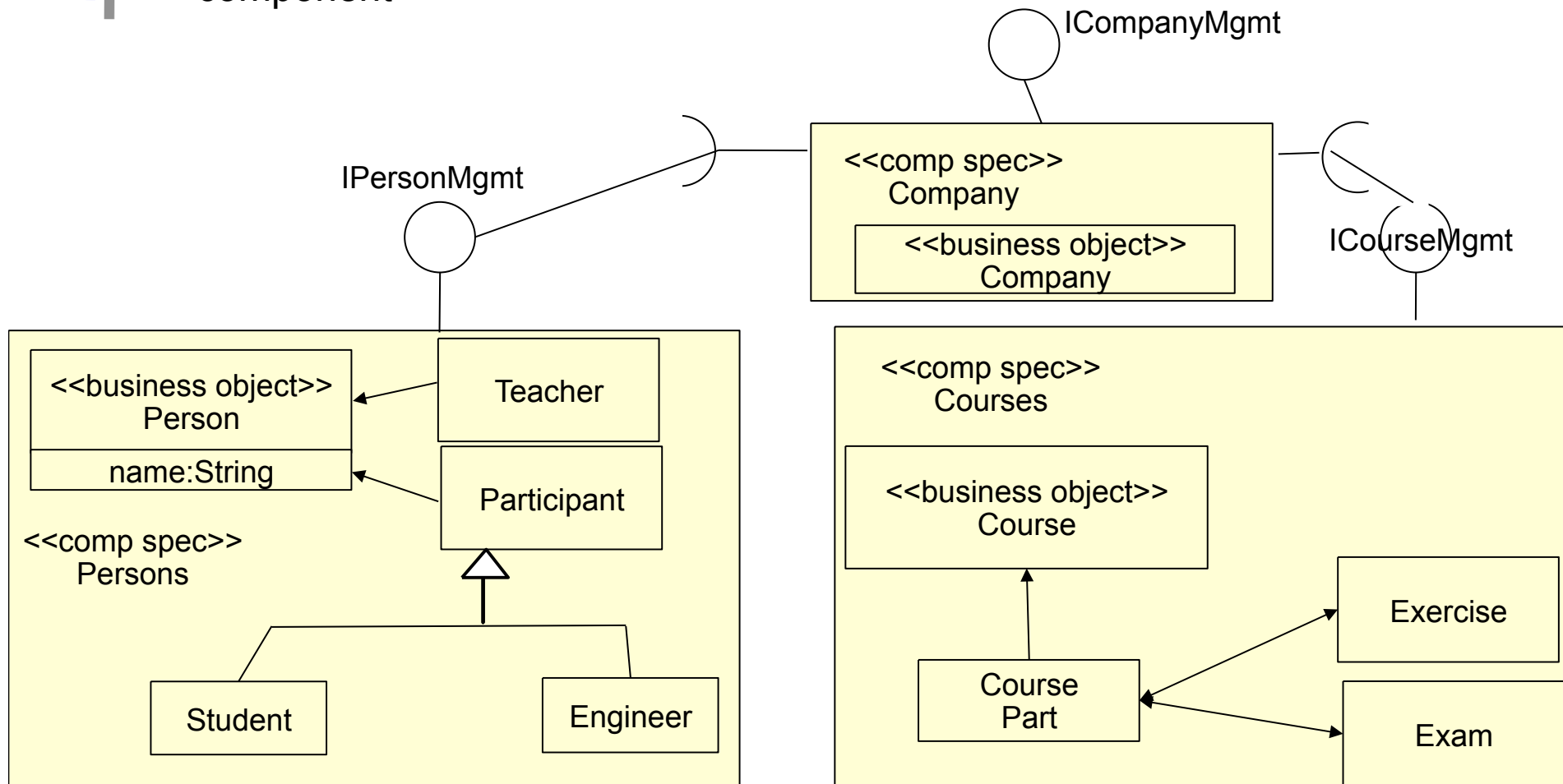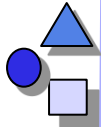
Component Identification

► The **component identification** subprocess attempts to

- Create a business object interface model from the domain model (still without methods)

- Attempts to group these interfaces to initial *system component specifications*

  - The grouping is done according to

    - *information hiding*: what should a component hide, so that it can easily be exchanged and the system can evolve?

    - *Reuse considerations:* which specifications of components are found in the component specification repository, so that they can be reused?

► There is a tension between business concepts, coming from the business domain (problem domain), and system components (solution domain). This gap should be bridged.

10.3.2 Component Interaction Analysis (Step 2) for Refinement of Interfacts

```
┌──────────────────────┐          ┌──────────────────────────┐
│   Business Object     │          │  Component Specifications │
│   Interface Model     │          │  and Architecture (0.1)   │
└──────────────────────┘          └──────────────────────────┘
            │                                   │
            ▼                                   ▼
┌─────────────────────────────────────────────────────────────────────────────┐
│        ╭──────────────────╮          ╭──────────────────╮    ┌──────────────┐ │
│        │  Add Operations  │          │   Architecture   │    │  Component   │ │
│        ╰──────────────────╯          │     Analysis     │    │ Interaction  │ │
│                 │                    ╰──────────────────╯    │  Analysis    │ │
│                 ▼                             │              └──────────────┘ │
│        ┌──────────────────┐          ╭──────────────────╮                     │
│        │ Business Object  │─────────▶│     Refine       │                     │
│        │     Model        │          │   Interfaces     │                     │
│        └──────────────────┘          ╰──────────────────╯                     │
│                 │                             │                               │
│                 │                             ▼                               │
│                 │                    ┌──────────────────────────┐             │
│                 │                    │  Component Specifications │             │
│                 │                    │  and Architecture (0.2)   │             │
│                 │                    └──────────────────────────┘             │
└─────────────────┼─────────────────────────────┼─────────────────────────────┘
                  ▼                              ▼
```
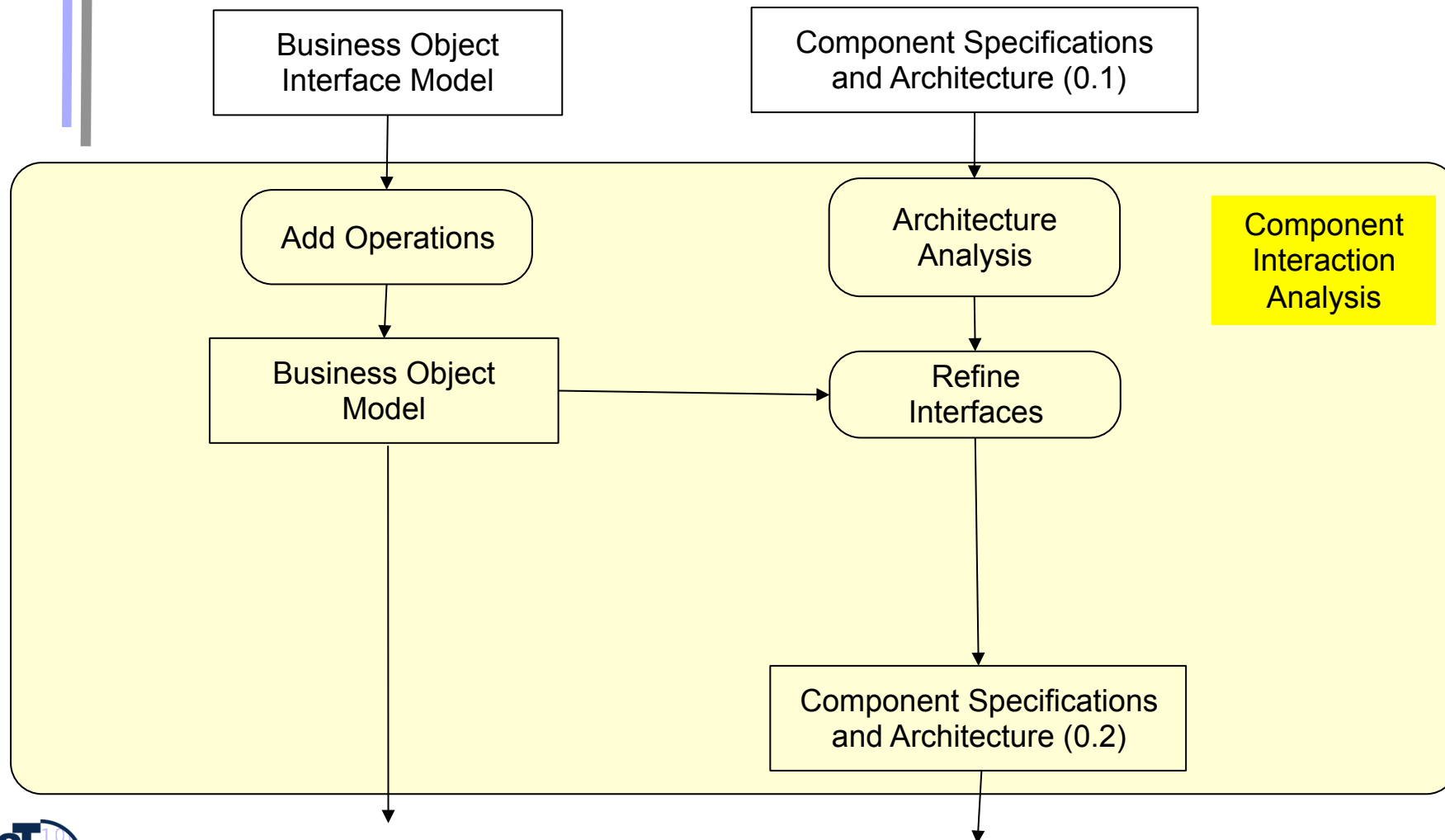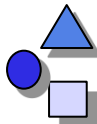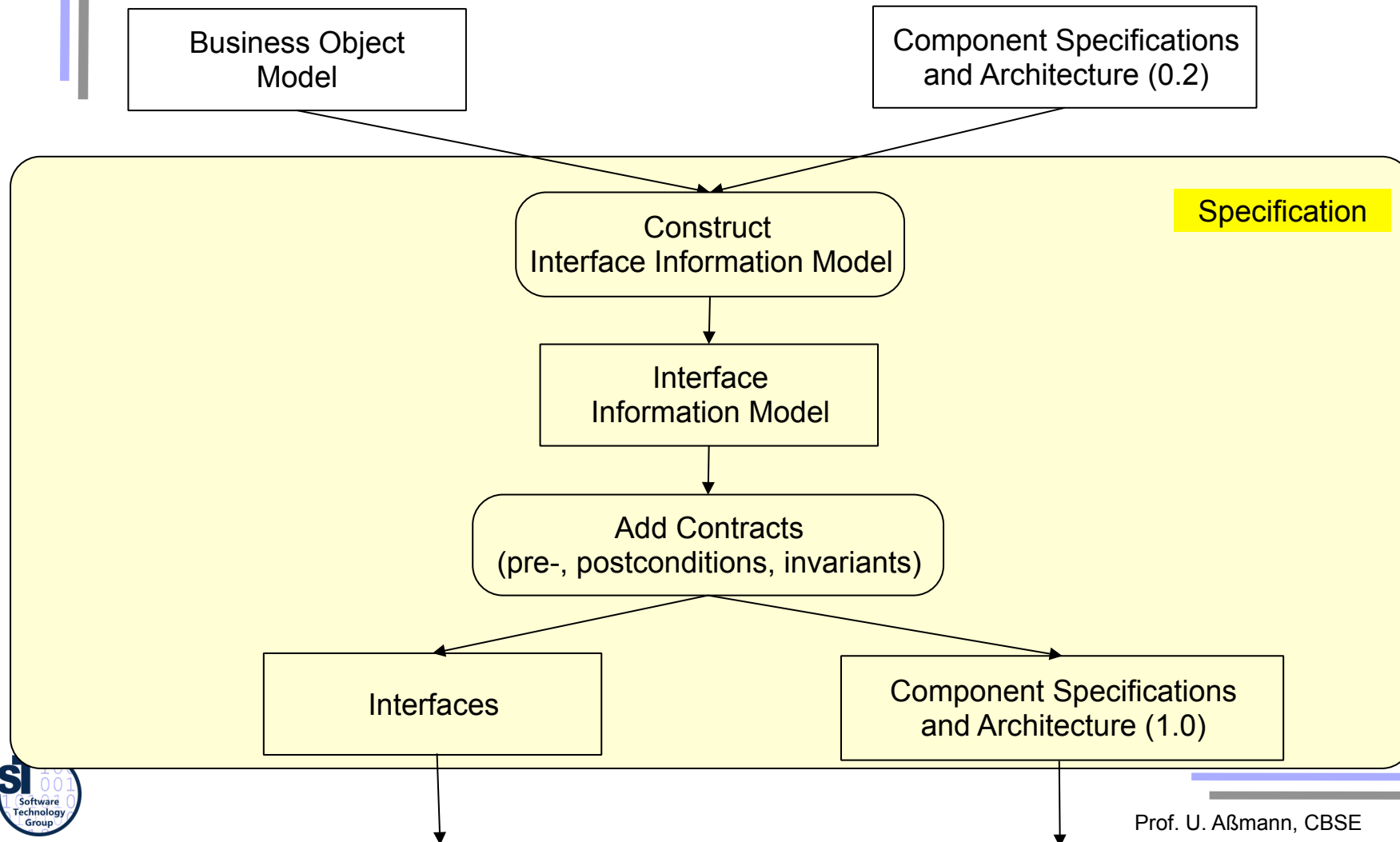
Component Interaction Analysis

► Is basically a refinement of the first stage

- Removing,

- Regrouping,

- Augmenting,

- Producing component specifications and wirings in a version 0.2

► Additionally, operations are added to business object interfaces

- And mapped to internal types.

10.3.3 Component Specification (Step 3)

➢ Enrich the interfaces with contracts

Business Object Model		Component Specifications and Architecture (0.2)

Construct Interface Information Model

Specification

Interface Information Model

Add Contracts (pre-, postconditions, invariants)

| Interfaces	Component Specifications and Architecture (1.0)

Component Specification (Step 3)

▶ Specification of declarative contracts for UML bobs in OCL

▶ Invariant construction:

   ▪ Evaluate business domain rules and integrity constraints

   ▪ Example:

```
context r: Course
-- a course can only be booked if it has been allocated in
   the company
inv:  r.bookable = r.allocation->notEmpty
```

▶ Pre/Postconditions for operations

   ▪ Can only be run on some state-based representation of the component

   ▪ Hence, the component must be modeled in an *interface information model*

   ▪ Or: be translated to implementation code (e.g. Java using an OCL2Java Compiler)
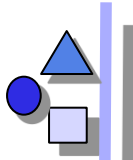
10.3.4. Provisioning (Realization, Implementation) (Step 4)

► **Provisioning selects component implementations for the specifications**

- Choosing a concrete implementation platform (EJB, CORBA, COM+, ...)
- Look up component implementations in implementation repositories
  - Write adapters if they don't fit exactly
- Program missing components
- Store component implementations and specifications in database for future reuse
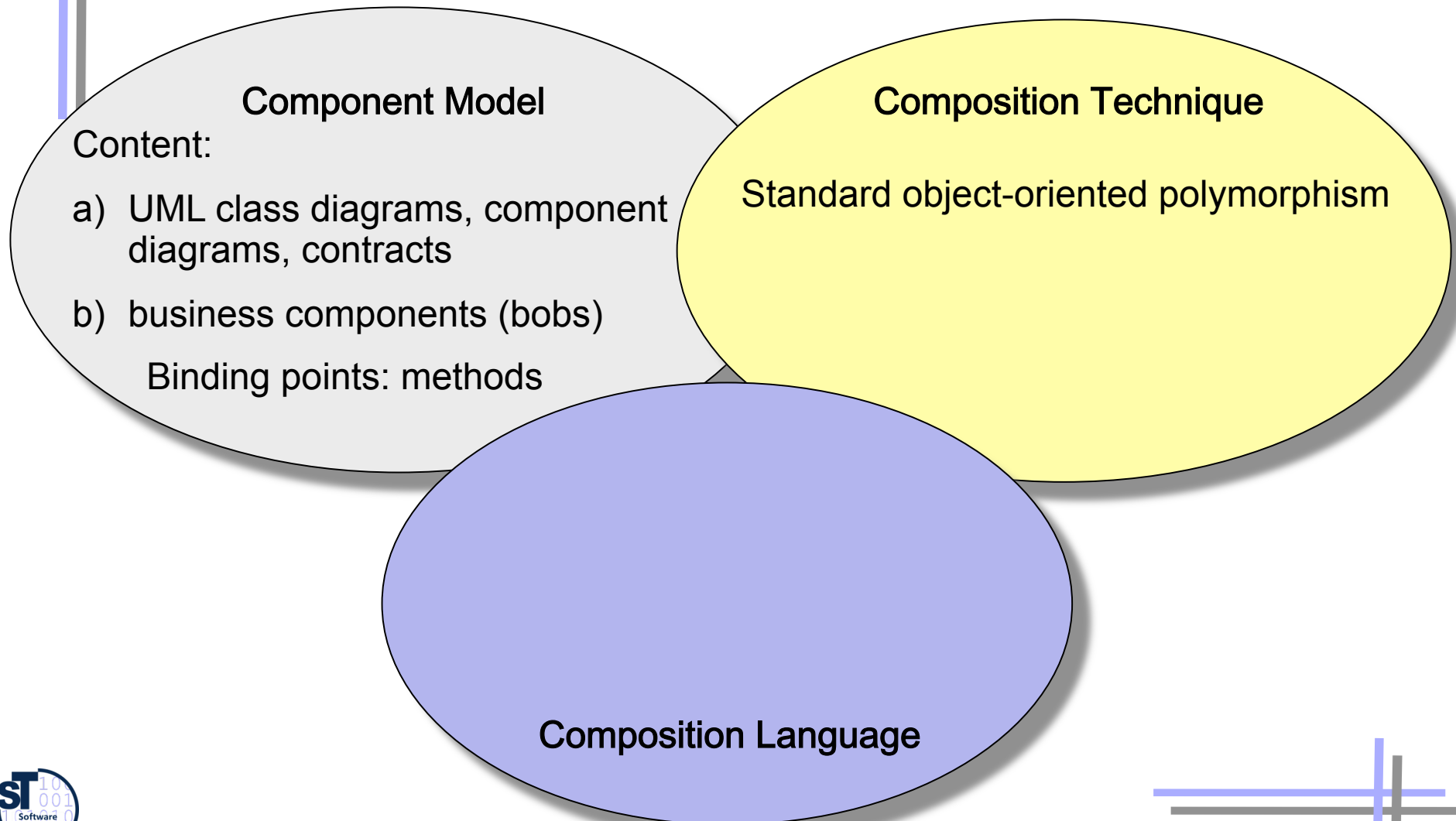
10.3.5 Assembly (Step 5)

► Puts together architecture, component specifications and implementations, existing components

   ▪ We will see more in the next lectures
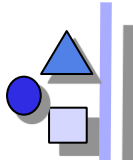
Weaknesses of Cheesman-Daniels Business Components

► No top-down decomposition of components, only bottom-up grouping from class diagrams

  ▪ part-of relationship is not really supported

► Reuse of components is attempted, but

  ▪ Finding components is not supported (see companion lecture)

    . Metadata

    . Facet-based classification

Cheesman-Daniels' Business Component Model as Composition System

Component Model

Content:

a) UML class diagrams, component diagrams, contracts

b) business components (bobs)

Binding points: methods

Composition Technique

Standard object-oriented polymorphism

Composition Language

The End